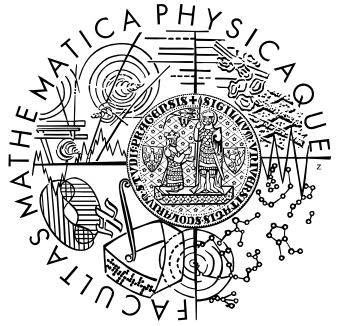
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Růžička

Particle systems

Kabinet software a výuky informatiky

Vedoucí bakalářské práce: RNDr. Josef Pelikán
Studijní program: informatika, programování
2009

„Neúspěch nás učí, že porážka se dá přežít. Neuspět není ostuda. Ostuda je bát se vstát a zkusit to znovu.“

Benjamin Barber

Dříve než přistoupím k vlastní práci, považuji za svou milou povinnost poděkovat tímto všem lidem, kteří mi pomohli s touto prací. Především však své rodině, za poskytnuté zázemí i důvěru po celou dobu studia a vedoucímu práce RNDr. Josefu Pelikánovi za podnětné rady a ochotné konzultace v době, kdy jsem je potřeboval.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne Martin Růžička

ANOTACE

Název práce: Particle systems
Autor: Martin Růžička
Katedra (ústav): Kabinet software a výuky informatiky
Vedoucí bakalářské práce: RNDr. Josef Pelikán
e-mail vedoucího: Josef.Pelikan@mff.cuni.cz

Abstrakt: Předložená práce se zabývá problematikou částicových systémů. Snaží se podat systém pro tvorbu, simulaci a vizualizaci částic. V úvodní části se věnuje návrhu fyzikálního modelu. Fyzikální model klade důraz zejména na rychlou a přitom uvěřitelně přesnou aproximaci fyzikálních zákonitostí. Poté se zabývá způsoby popisu chování částic. Pro co největší flexibilitu je použito skriptovacího rozhraní. Závěr práce potom řeší vlastní vizualizaci koncových částic. Ta je realizována prostřednictvím navržené nadstavby nad grafickou knihovnu OpenGL. Výsledky práce je možné si prohlédnout v přiložené aplikaci demonstrující možnosti navrženého částicového systému.

Klíčová slova: skriptování, fyzikální simulace, částice, OpenGL

Title : Particle systems
Author : Martin Růžička
Department : Department of Software and Computer Science Education
Supervisor : RNDr. Josef Pelikán
Supervisor's e-mail address : Josef.Pelikan@mff.cuni.cz

Abstract : The presented work deals with the issue of particles. It aims to offer a system for particle creation, simulation and visualization. The introductory part of the paper is devoted to design a physical model. The physical model particularly stresses a fast and at the same time plausible approximation of physical laws. Afterwards, it deals with methods of describing the particle behavior. In order to achieve the highest possible flexibility, a scripting interface is used. The conclusion then describes the visualization of the final

particles itself. It is realized using the designed extension of the OpenGL graphic library. The results of the work can be seen in an enclosed application which demonstrates the possibilities of the designed particle systems.

Keywords : scripts, physical simulation, particles, OpenGL

OBSAH

1 ÚVOD

Částice jsou všude kolem nás. S trochou nadsázky je můžeme považovat za základní stavební kameny veškerých objektů ve vesmíru. Z pohledu mikrosvěta to mohou být atomy nebo molekuly. V makrosvětě potom větší celky chovající se navenek jako jednotná jednoduchá tělesa. V obou případech nám však částice poslouží jako použitelný základ pro další práci. Tento pohled na částici, jako na základní prvek všech objektů, je poměrně zajímavý a přináší některé významné vlastnosti. Částice jsou například dobře a jednoduše simulovatelné objekty, u kterých si vystačíme pouze s několika základními atributy. Také fyzikální model odpovídá poměrně přesně skutečnosti. Vždyť v reálném světě nejsou tělesa nic jiného než shluky částic vzájemně spojených nejrůznějšími vazbami. Tento přístup s sebou přináší ale i určité potíže. Hlavní nevýhodou je především potřeba velkého množství částic. Problematické jsou i vazby mezi jednotlivými částicemi pro vytvoření komplexnějších pevných těles. Z uvedených důvodů jsou proto částice obvykle používány jen k simulaci nerigidních fyzikálních objektů. Ukážeme však, že je lze mnohdy dobře využít i k tvorbě složitějších pevných těles.

1.1 Motivace

Počítačová grafika dnes klade velký důraz na co nejvěrnější způsob zobrazování. Snahou je přiblížit se co nejvíce fotorealistickému vzhledu. Lépe se pak uplatní v nejrůznějších oborech jako je například filmový průmysl, hry a mnoho dalších. Zatímco někde vystačíme s jednotlivými statickými renderovanými snímky, jinde často požadujeme souvislé pohyblivé sekvence. V takových případech už obvykle nepostačí pouze realistický vzhled, ale pro dodání uvěřitelné scény je zapotřebí i jiných faktorů – například fyziky. Fyzika objektům dodává další potřebné vlastnosti, které významným způsobem dotvářejí realistický dojem z výsledné scény. Pro celkový dojem je tedy potřebný nejen realistický vzhled, ale i dobrý fyzikální model.

1.2 Cíle práce

Ačkoliv si tato práce neklade za cíl realistické zobrazování částic ani dokonalou fyzikální simulaci, snaží se předložit použitelný a dobře rozšiřitelný systém pro jednoduchou a rychlou tvorbu částicových efektů zakládajících se na korektních fyzikálních pravidlech, které umožní vytvářet zajímavé real-time efekty. Pomocí navrženého systému bude možné

definovat, simulovat a zobrazovat efekty složené z částic. Především pak bude kladen důraz na tvorbu a popis částic, jejich fyzikální simulaci a vizualizaci. Neodmyslitelnou součástí tohoto projektu je i přiložená aplikace, která ukazuje praktické výsledky práce. Za všechny jmenujme například efekty kouře, kyvadla nebo vlající vlajky.

2 FYZIKÁLNÍ MODEL

Hlavním cílem této kapitoly je volba vhodného fyzikálního modelu. Fyzikální model přímo ovlivňuje kvalitu, rychlost a možnosti našich budoucích částicových simulací. Jeho volba závisí především na cílové skupině simulací, které budeme od modelu vyžadovat.

2.1 Klasická mechanika

V této práci se zaměříme především na simulaci makroskopických těles, kterými se zabývá hlavně klasická mechanika. Klasická mechanika popisuje fyzikální zákonitosti makro částic, které se v porovnání s rychlostí světla pohybují zanedbatelně pomalu. Klasickou mechaniku definují zejména Newtonovy pohybové zákony, které zde proto v krátkosti zopakujeme.

2.1.1 Newtonovy pohybové zákony

1) Zákon setrvačnosti: „Jestliže na těleso nepůsobí žádné vnější síly nebo výslednice sil je nulová, pak těleso setrvává v klidu nebo v rovnoměrném přímočarém pohybu.“
2) Zákon síly: „Jestliže na těleso působí síla, pak se těleso pohybuje se zrychlením, které je přímo úměrné působící síle a nepřímo úměrné hmotnosti tělesa.“
3) Zákon akce a reakce: „Proti každé akci vždy působí stejná reakce.“ nebo také: „Vzájemná působení dvou těles jsou vždy stejně velká a míří na opačné strany.“

2.2 Výpočetní model

Další rozhodnutí, které v souvislosti s fyzikálním modelem musíme učinit, je způsob, jakým budeme vlastní simulace počítat. Nabízí se několik běžně používaných variant. Každá s poněkud odlišnými možnostmi a cílovým zaměřením. Než se pro nějaký konkrétní způsob rozhodneme, představme si nejprve uvažované výpočetní modely.

2.2.1 Analytický integrační model

Analytický integrační model se vyznačuje dokonalou přesností výpočtu v libovolném časovém okamžiku simulace. Velkým problémem však zůstává jeho složitost. Z uvedeného důvodu se analytické řešení používá pouze pro menší a jednodušší výpočty.

Pro lepší představu problému uvažme následující situaci z práce [1]. Mějme částici s nulovou počáteční rychlostí a dráhou. Částice začne padat volným pádem a ptáme se na dráhu, kterou urazí za čas t. Jde o zcela jednoduchou situaci, která se však v případě analytického výpočtu rozvine v poměrně složitý a výpočetně náročný postup.

Podle 2. Newtonova zákona platí

$$F = m \cdot a$$

Na těleso působí gravitační síla, tedy

$$F = m \cdot g$$

Okamžité zrychlení spočteme jako 2. derivaci dráhy podle času

$$a = s^{(2)}$$

Dosazením do rovnice dostáváme

$$s^{(2)} \cdot m = g \cdot m$$

Prvním zintegrováním podle času vychází

$$s^{(1)} = k + g \cdot t$$

A po druhém zintegrování konečně

$$s = c + k \cdot t + \frac{1}{2} \cdot g \cdot t^2$$

Tím jsme vyjádřili přesný vzoreček pro výpočet uražené dráhy v daném čase.

Můžeme namítat, že ačkoliv je vlastní odvození složité, pro simulaci vystačíme pouze s výslednými vzorečky. Co když se ale částice v průběhu volného pádu přiblíží k jiné částici, která ovlivní její rychlost, nebo dojde ke střetu s jiným tělesem a tím k vychýlení z původní dráhy letu? Pro řešení uvedených případů by bylo potřeba rovnici rozšířit o nové proměnné. Budeme-li však simulovat větší množství objektů současně, stane se taková soustava rovnic prakticky neřešitelná. Z uvedeného příkladu je zřejmé, že analytické řešení je pro naše účely nevhodné.

2.2.2 Numerický integrační model

Numerický integrační model aproximuje průběh původní funkce. Metody simulace založené na numerickém integračním modelu nazýváme diskrétními. Diskrétní simulace vzorkuje čas a počítá stav simulace rekurentně z předchozího známého stavu. Hlavní výhodou je jednoduchost a rychlost výpočtu v porovnání s analytickým způsobem řešení. Rekurentní způsob aproximace však zároveň vede ke vzniku a akumulaci nepřesností v průběhu simulace.

Cílem této práce však není dodat dokonale věrný simulační model a proto si s numerickým integračním modelem dobře vystačíme. Zbývá rozhodnout, jakým způsobem budeme původní funkci aproximovat. Nabízí se hned několik možností, většina z nich přímo vychází z Taylorovy řady [2], kterou zde proto v krátkosti zopakujeme.

2.2.3 Taylorova řada

Za předpokladu, že funkce f má v bodě *a* derivace všech řádů, lze tuto funkci vyjádřit jako

$$f(x)=\sum_{k=0}^{\infty}\frac{f^{(k)}(a)}{k!}\cdot(x-a)^k$$

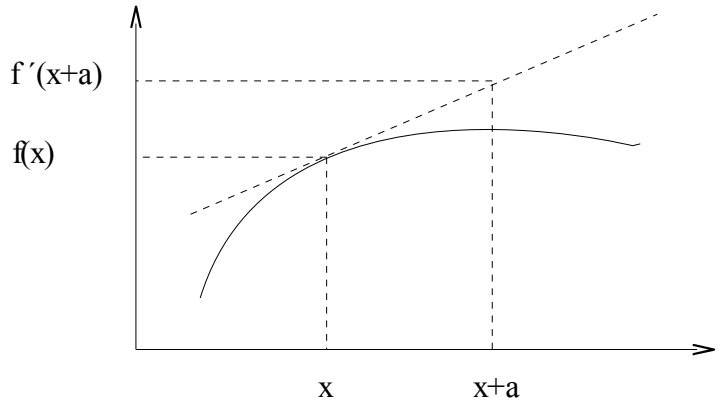
Přesnost aproximace funkce v daném bodě tedy závisí na počtu členů Taylorovy řady. Z uvedeného vztahu je zřejmé, že s rostoucími členy této mocninné řady klesá rychle jejich vliv na výslednou hodnotu funkce. Proto pro přibližnou aproximaci funkce obvykle stačí spočítat několik málo prvních členů rozvoje. Tím získáváme tzv. Taylorův polynom n-tého řádu, kde n určuje počet prvních členů Taylorovy řady.

2.2.4 Eulerova metoda

Jde o nejjednodušší, nejrychlejší ale také nejméně přesnou metodu aproximace vycházející z Taylorovy řady. Metoda využívá Taylorův polynom 1. řádu. Formálně vyjádřeno vztahem

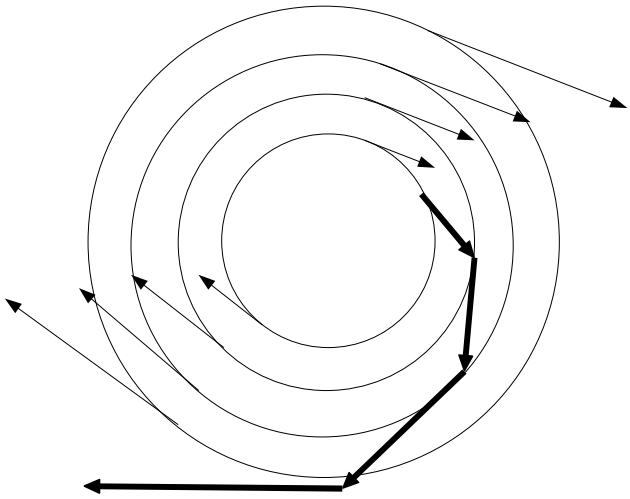
$$f'(x)=\sum_{k=0}^{1}\frac{f^{(k)}(a)}{k!}\cdot(x-a)^k=f(a)+f^{(1)}(a)\cdot(x-a)$$

Eulerova metoda tedy průběh funkce aproximuje tečným vektorem. Situaci názorně ukazuje přikládaný obrázek.



Aproximace funkce Eulerovou metodou

Z přiložené ilustrace je zřejmé, že Eulerova metoda je nepřesná zejména při nelineárním průběhu funkce. Uvažme například situaci, kdy se částice pohybuje po kružnici. Eulerova metoda v takovém případě počítá tečný pohyb částice v daném bodě a podle časového intervalu částici v tomto směru posune. Jak je z obrázku níže patrné, výsledek začne velmi rychle divergovat od očekávaných hodnot. Tento problém můžeme částečně kompenzovat snížením časového kroku simulace. Velikost odchylky aproximované funkce od hodnot původní funkce je totiž přímo úměrná druhé mocnině simulačního kroku [3]. Ani tím se však výrazné divergenci v pozdější fázi simulace nevyhneme a dochází k tzv. numerické nestabilitě [4]. Situaci názorně ukazuje následující obrázek převzatý z práce [1].



2.2.5 Další metody

Vyšší řády Taylorova polynomu používají například tzv. Midpoint method (počítá polynom 2. řádu), Runge Kutta (3, 4, 5. řádu) a další. Tyto metody významným způsobem zvyšují přesnost aproximace funkce, ale zároveň snižují rychlost výpočtu. Používané jsou také vícekrokové metody, využívající více předchozích stavů k určení nového stavu. Jejich nevýhodou je nižší rychlost a potřeba uchovávat historii dřívějších stavů.

2.2.6 Volba vhodné metody

Stojíme před klíčovým rozhodnutím. Nabízí se nám na jedné straně rychlý ale nepřesný výpočetní model, na straně druhé přesnější, pomalejší a složitější model. Který tedy zvolit?

S ohledem na vytyčené cíle práce byla zvolena Eulerova metoda. Tato metoda umožní simulaci většího množství částic. V situacích, kdy potom budeme pracovat s menším množstvím částic, budeme její nepřesnosti kompenzovat častější aproximací funkce na menších intervalech. Nabízí se i možnost adaptivní volby simulačního kroku podle velikosti chybové odchylky, jak to popisuje autor v [5]. Tím docílíme možnosti simulovat jak velké množství částic s menší přesností, tak menší množství částic s přesností větší.

3 ODVOZENÍ POUŽITÝCH FYZIKÁLNÍCH VZTAHŮ

Fyzikální model máme tedy zvolen a můžeme přistoupit k vlastnímu odvození fyzikálních vzorečků, které budeme v průběhu simulace používat. Pro naprostou většinu simulací jistě budeme potřebovat počítat rychlost, zrychlení, působící síly a v neposlední řadě i uraženou vzdálenost. Proto se na tyto fyzikální veličiny podívejme blíže. Uvedené vzorečky vycházejí z informací uvedených v [6] a [7].

3.1 Rychlost

Rychlost je vektorová fyzikální veličina vypovídající o směru a uražené dráze tělesa za jednotku času.

Nechť $\vec{f}$ je vektorová funkce jedné proměnné, vyjadřující závislost polohy na čase. Okamžitou rychlost spočítáme jako derivaci uražené dráhy podle času.

$$\vec{v} = \lim_{(t_1 \to t_2)} \frac{\vec{f}(t_1) - \vec{f}(t_2)}{t_1 - t_2} = \frac{d\,\vec{f}(t)}{dt} = \frac{d\,\vec{s}}{dt}$$

3.2 Zrychlení

Zrychlení je, podobně jako rychlost, vektorová fyzikální veličina určující změnu a směr rychlosti za jednotku času.

Průměrné zrychlení na časovém intervalu t_1, t_2 je dáno vztahem

$$\vec{a}_{p_{t_1,t_2}} = \frac{\vec{v}(t_1) - \vec{v}(t_2)}{t_1 - t_2}$$

Okamžité zrychlení určíme jako $\vec{a} = \lim_{(t_1 \to t_2)} \frac{\vec{v}(t_1) - \vec{v}(t_2)}{t_1 - t_2} = \frac{d\,\vec{v}(t)}{dt} = \frac{d\,\vec{s}^{(2)}}{dt^2}$, tedy jako 2. derivaci uražené dráhy podle času.

3.3 Síla

Síla je fyzikální vektorová veličina vyjadřující míru vzájemného působení těles.

Podle 2. Newtonova zákonu je definována jako změna hybnosti za jednotku času.

$$\vec{F} = \frac{d\,\vec{P}}{dt}$$

V případě, že je změna hmotnosti během pohybu zanedbatelně malá, lze potom vztah zjednodušit na

$$\vec{F} = m \cdot \frac{d\,\vec{v}}{dt} = m \cdot \vec{a}$$

Na částice v našich simulacích bude obvykle působit více sil různého původu zároveň. Uveďme proto alespoň ty nejčastější – jsou to zejména pružiny, tlumiče a gravitační síla.

3.3.1 Pružiny

V simulacích budeme pracovat pouze s lineárními pružinami. Síla, kterou taková pružina působí, je dána vztahem

$\vec{F}_s = -k_s \cdot (L - r) \cdot |\vec{L}|$, kde L je délka pružiny,

$|\vec{L}|$ je normalizovaný vektor pružiny, r je klidová délka pružiny a k_s je tuhost pružiny.

3.3.2 Tlumič

Proti vznikajícím silám působí síly opačné, které tyto síly redukují. Souhrnně je označujeme jako tlumiče. Sílu tlumiče lze vyjádřit vztahem

$\vec{F}_d = k_d \cdot (\vec{v}_1 - \vec{v}_2)$, kde k_d je tlumící faktor a

$\vec{v}_1, \vec{v}_2$ jsou rychlosti těles, na které tlumič působí.

3.3.3 Gravitační síla

Gravitační síly působí vždy přitažlivě. Pro naše simulace budeme typicky předpokládat homogenní gravitační pole, kterým je například i povrch Země, to lze vyjádřit vztahem

$\vec{F}_g = m \cdot \vec{g}$, kde $\vec{g}$ je gravitační zrychlení působící ve směru do středu tělesa, stojícího za vznikem gravitačního pole.

3.4 Výsledná pozice, rychlost a síla

Uvedli jsme seznam fyzikálních veličin, které pro simulace budeme používat. Zbývá ukázat, jakým způsobem vše zkombinujeme dohromady a určíme výslednou sílu, rychlost a pozici simulovaných těles.

Ze 3. Newtonova zákona vyplývá, že velikost sil, kterými na sebe působí vzájemně dvě tělesa, je stejná a má pouze opačný směr. Toho můžeme využít a snížit tak počet prováděných výpočtů a tím urychlit simulaci.

Další důležité zjednodušení nám umožní tzv. princip superpozice. Ten říká, že síly, působící na těleso jsou vzájemně nezávislé a lze je vektorově sčítat. Z toho přímo plyne, že nám stačí vyhodnotit pouze výslednou sílu působící na těleso a podle ní dopočítat uraženou vzdálenost.

Nechť $\vec{F}_1, ..., \vec{F}_n$ jsou všechny síly působící na těleso v čase t-1. Výslednici těchto sil potom spočítáme podle vztahu

$$\vec{F} = \sum_{i=1}^{n} \vec{F}_i$$

Podle 2. Newtonova zákona pro zrychlení v čase t-1 platí

$$\vec{a} = \frac{\vec{F}}{m}$$

Eulerovou metodou spočítáme přírůstek rychlosti za čas t_Δ

$$\boxed{\vec{v}_\Delta = (\frac{\vec{F}}{m}) \cdot t_\Delta}$$, kde t_Δ je čas uplynulý mezi předchozím a aktuálním simulačním krokem.

Výslednou rychlost v čase t potom určíme v souladu s Newtonovým zákonem setrvačnosti jako součet rychlosti v čase t-1 a nově získané rychlosti za čas t_Δ

$$\vec{v}_t = \vec{v}_{t-1} + \vec{v}_\Delta$$

Nakonec , když známe novou rychlost zbývá dopočítat uraženou vzdálenost

$\boxed{\vec{s}_\Delta = \vec{v}_t \cdot t_\Delta}$, kde $\vec{s}_\Delta$ značí uraženou dráhu za čas t_Δ

Tím máme odvozeny všechny důležité vzorečky, které pro fyzikální simulaci budeme využívat.

4 FYZIKÁLNÍ SIMULACE

V předchozí části práce jsme zvolili fyzikální model, uvedli a vysvětlili vzorečky a můžeme tak přistoupit k vlastnímu návrhu simulace. Vyjdeme z jednoduchého fyzikálně simulovatelného objektu. Tento objekt bude obsahovat všechny důležité atributy potřebné pro základní fyzikální simulaci a z tohoto objektu potom budeme odvozovat specializovanější třídy. Co ale do tohoto základního objektu ukládat a co naopak ne? Jak budeme provádět vlastní simulaci takových objektů? Jaké potomky z navrženého objektu budeme vytvářet? Na položené otázky se pokusíme odpovědět v průběhu této kapitoly.

4.1 Volba základní sady atributů

Bázová třída fyzikálních objektů by měla být co nejmenší, abychom odvozené třídy zbytečně nezatěžovali vlastnostmi, které ani nebudeme potřebovat a šetřili tak paměť počítače i čas procesoru. Současně by ale tento základ měl obsahovat všechny atributy nutné pro základní fyzikální simulace abychom mohli u potomků tento základ pohodlně využívat a pouze jej rozšiřovat o další fyzikální možnosti.

Abychom mohli jednoznačně popsat, kde se v třírozměrném prostoru objekt nachází, budeme potřebovat uchovávat jeho pozici. Pro pohodlný a intuitivní popis pozice použijeme kartézskou soustavu souřadnic, kde pozici objektu na každé z os bude popisovat zvláštní atribut. Pro prakticky libovolnou simulaci budeme také potřebovat informaci o rychlosti objektu. Jak jsme řekli v předchozí kapitole, rychlost je vektorová veličina a proto ji budeme reprezentovat vektorem vyjadřujícím směr a velikost. S rychlostí obvykle souvisí i zrychlení, které proto také zahrneme. Potřebovat budeme i hmotnost objektu. Na objekt dále budou působit nejrůznější síly, jako například gravitační, síla pružiny, odpor vzduchu, tření a další. Výslednici těchto sil budeme uchovávat v tzv. akumulátoru sil, který do částice také zahrneme. Nabízí se plno dalších vlastností, které by pro simulaci občas mohly být užitečné, například hustota, objem, materiál objektu a mnohé jiné. Jejich využití však nebude zdaleka tak časté jako dříve jmenované, proto je do bázové třídy fyzikálních objektů zahrnovat nebudeme. Pouze v případě, že je pro nějakou simulaci budeme skutečně potřebovat, je přidáme do některého specializovanějšího potomka této třídy.

4.2 Simulace objektu

Navrhli jsme základní sadu atributů fyzikálních objektů a můžeme přistoupit k jejich vlastní simulaci.

Pozice objektu bude jediná vlastnost, která se navenek bude projevovat a zvolená sada fyzikálních atributů tuto pozici bude ovlivňovat. Pozici objektu budeme vzhledem k diskrétnímu způsobu simulace vyhodnocovat vždy v předem určených časových intervalech. Čím menší časový úsek budeme simulovat, tím bude výsledek přesnější. S rostoucím počtem prováděných simulačních kroků za jednotku času zároveň také poroste vytížení procesoru vedoucí k celkovému zpomalování programu. Proto je důležité mít tento simulační krok nastavitelný a v závislosti na požadavcích jednotlivých simulací ho moci měnit.

4.2.1 Simulační funkce

Budeme tedy potřebovat funkci pracující s fyzikálním objektem a časovým krokem, na kterém objekt simulujeme. Funkce vyhodnotí aktuální rychlost objektu a v závislosti na velikosti časového kroku objekt o uraženou vzdálenost posune.

Rychlost objektu budeme počítat podle vztahů popsaných ve 3. kapitole.

$\vec{v}_t = \vec{v}_{t-1} + \vec{v}_\Delta$, kde

$\vec{v}_t$ je nová aktuální rychlost v čase t

$\vec{v}_{t-1}$ je rychlost v čase t-1

$\vec{v}_\Delta$ udává přírůstek rychlosti za čas t_Δ , kterou určíme

vztahem $\vec{v}_\Delta = (\frac{\vec{F}}{m} + \vec{a}_{const}) \cdot t_\Delta$

$\vec{F}$ je výslednice sil uložená v akumulátoru sil působících na těleso

$\vec{a}_{const}$ je konstantní zrychlení objektu

t_Δ je časový krok.

Výslednou pozici objektu určíme jako

$\vec{pos}\,OBJ = \vec{pos}\,OBJ + \vec{s}_\Delta$, kde

$\vec{s}_\Delta = \vec{v}_t \cdot t_\Delta$ je uražená dráha za čas t_Δ

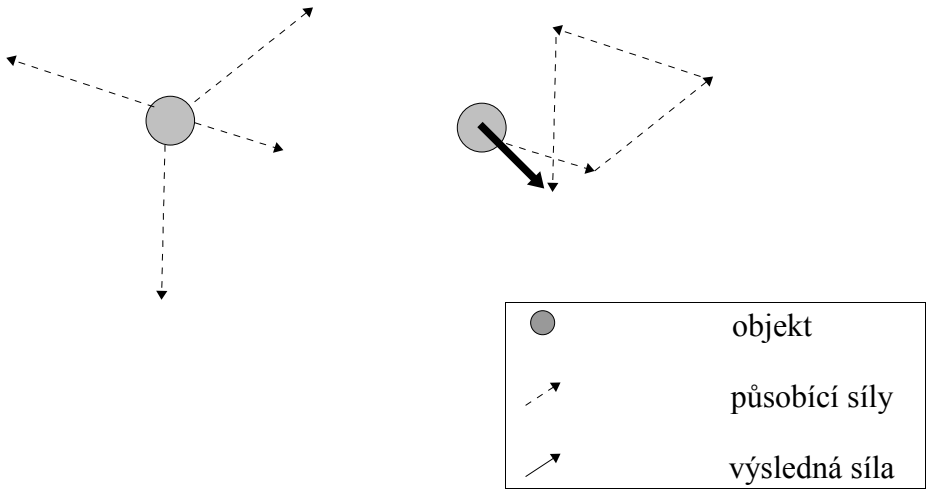
Pseudokód výsledné funkce pak může vypadat například následovně:

```
SIMULATE(object, timestep)
{
   object.velocity+=((object.final_force/object.weight)
                    +object.const_acceleration)*timestep

   object.position+=  object.velocity*timestep

   object.final_force.clear()
}
```

Zbývá objasnit způsob přidávání působících sil do akumulátoru. Z dříve zmíněného principu superpozice víme, že síly působící na těleso jsou vzájemně nezávislé a lze je vektorově sčítat. V akumulátoru sil nám tedy stačí sčítat síly působící na těleso na daném časovém intervale a výslednou sílu na závěr podle uvedených vzorečků převést na rychlost.



Sčítání a vyhodnocení působících sil v akumulátoru

5 ČÁSTICE

V minulé kapitole jsme navrhli a popsali simulaci základního fyzikálního objektu. V této části práce přejdeme k tvorbě specializovanějších tříd. Předně musíme sestrojit objekty reprezentující částice, na jejichž simulaci se celá práce zaměřuje. Ke tvorbě částicových simulací budeme potřebovat doplnit navrženou základní fyzikální třídu o nové atributy, které každá částice bude obsahovat. Jde především o životnost částice, po jejímž uplynutí částice zanikne. Třídu dále rozšíříme o ukazatel na speciální objekt, který budeme používat pro případné dodatečné informace o částici. Tím dostáváme dobrý základ pro simulaci částic. Základní třídu pro tvorbu částic budeme v následujícím textu občas značit jako BASIC_PARTICLE.

5.1 Rozdělení částic

Pouze se základním druhem částice bychom si ve složitějších simulací jen velmi těžko vystačili a proto budeme používat dva druhy částic - emitory a koncové částice. Koncová částice obsahuje výše popsanou sadu atributů umožňující její fyzikální simulaci, dále ji budeme označovat z angličtiny jako PARTICLE. Emitor je potom částice rozšířená navíc o možnost tvorby nových částic. Označovat ji někdy budeme jako EMITTOR.

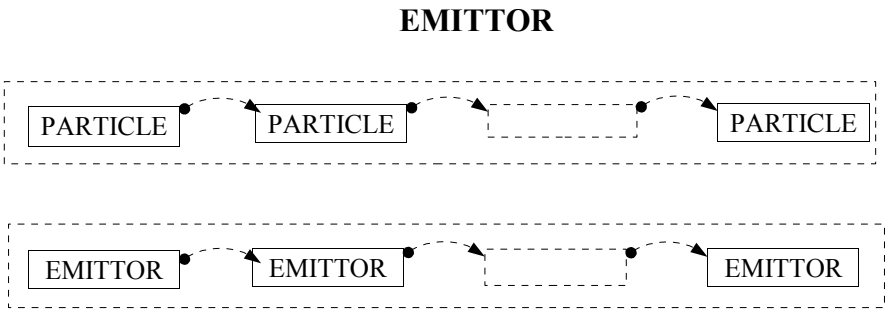
Na první pohled se může zdát zahrnutí koncových částic jako zbytečné - vždyť emitor zvládne vše co koncová částice a k tomu i něco navíc. Zde je potřeba si uvědomit, že rozšíření možností u emitorů s sebou přináší také zvýšené systémové nároky. Často si navíc s koncovými částicemi pohodlně vystačíme a proto je zbytečné platit za funkce, které nutně nepotřebujeme.

5.2 Uložení částic

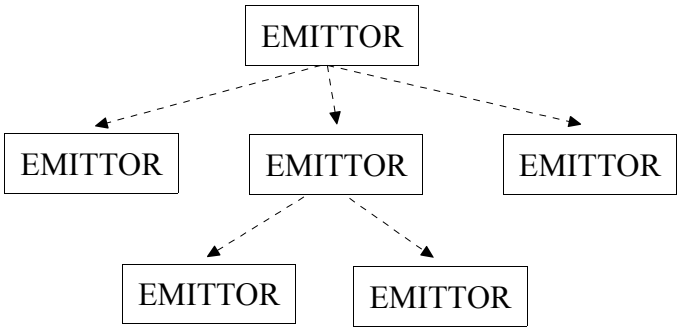
Částice budeme potřebovat uchovávat ve vhodných datových strukturách. Tyto struktury by nám s nimi měli umožnit příjemnou a efektivní práci. Co konkrétně však od takové struktury budeme vyžadovat?

V situacích, kdy uplyne životnost částice, bude potřeba částici z datové struktury odebrat. Existence emitorů bude zase vyžadovat možnost přidávání nových částic v průběhu simulace. Nakonec, abychom mohli s jednotlivými částicemi pracovat, budeme potřebovat i sekvenční přístup k částicím. Uvedená kritéria vedla k následujícímu návrhu.

Každý emitor obsahuje seznam ukazatelů na své potomky. Seznam potomků je přístupný prostřednictvím speciálních funkcí a metod. Potomkem může být buď PARTICLE nebo EMITTOR částice. Pro přehlednější a lépe rozšiřitelný návrh jsou udržovány dva seznamy potomků – jeden pro koncové částice a jeden pro emitory. Vzniká tak stromová hierarchická struktura, kde každá částice (s výjimkou emitoru v kořenu stromu, který dále budeme označovat řídící emitor) má svého rodiče. Rodičem je emitor, který částici vytvořil. Potomkem může být libovolná částice. Do seznamů lze k částicím jak přistupovat, tak i rychle přidávat a odebírat částice. Situaci ilustrují následující schémata.



Seznam potomků udržovaný v každém emitoru



Hierarchie emitorů

6 SKRIPTOVÁNÍ

V první kapitole jsme naznačili, že počet simulovaných částic bude mnohdy poměrně vysoký. Navíc na jednotlivé částice dále budeme často klást různé speciální požadavky. Bylo by tedy vhodné mít nějaký systém pro popis vlastností jednotlivých částic. Co všechno od takového systému budeme potřebovat? V první řadě to budou co nejflexibilnější možnosti nastavení atributů částic. Navržený systém by také měl být co nejrychlejší. Výhodou by byla i jednoduchost a přitom univerzálnost systému.

6.1 Proč skriptování

S ohledem na uvedené aspekty bylo použito skriptování. Pomocí skriptů lze jednoduše popisovat vlastnosti částic, a přitom nám skripty poskytují i požadovanou volnost. Uvnitř skriptů můžeme provádět nejrůznější výpočty, pomocí kterých pak definujeme vlastnosti částic. Jak je to ale s rychlostí takových skriptů? Skripty jsou za běhu interpretované programy a jsou proto pomalejší než kompilovaný kód. Proto se pokusíme zvolit co nejrychlejší skriptovací jazyk a používat ho budeme výhradně k vykonávání nejnutnějších úkonů. Zbylou část zodpovědnosti přenecháme kompilovanému kódu psaném v C++. Toto vedlo k následujícímu návrhu.

Skripty umožňují nastavení atributů částic. Definované vlastnosti jsou potom předány C++ programu, který částice požadovaných vlastností vytvoří, uloží do svých datových struktur a dále s nimi výhradně pracuje.

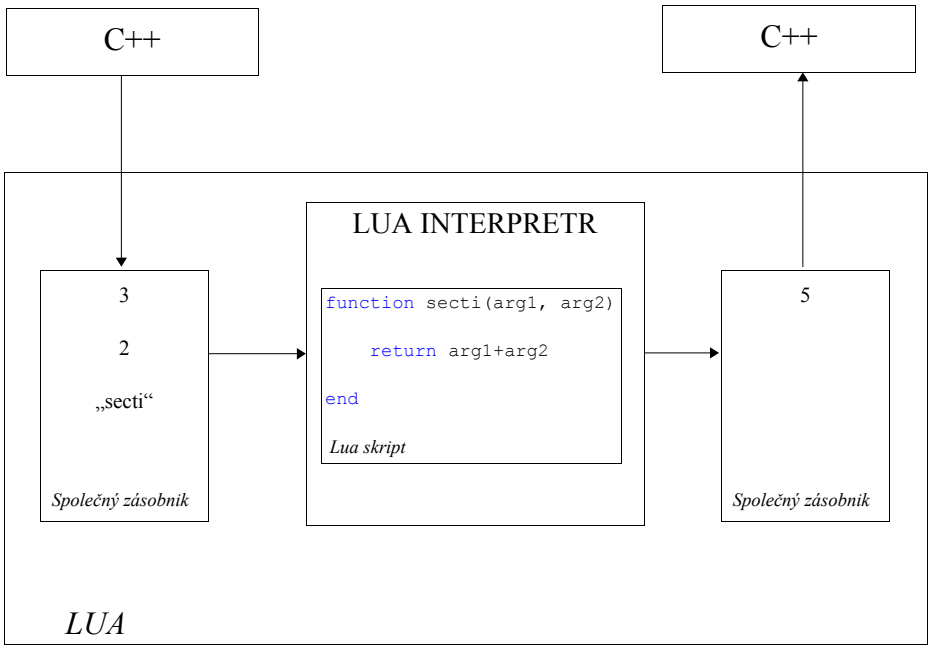
Schéma použití skriptování: program si vyžádá nové částice, skript částice definuje a jejich popis vrací zpět programu, který částice předepsaných vlastností vytvoří.

6.2 LUA skriptování

Dříve než přistoupíme k návrhu systému skriptů objasněme si, jak skripty blíže fungují. Skripty jsou za běhu interpretované programy. Proto je při změně není třeba znovu překládat, což umožňuje rychle a pohodlně tyto programy modifikovat a vytvářet. Ze stejného důvodu jsou ovšem také pomalejší než klasický kompilovaný kód. V programu byla jako skriptovací jazyk zvolena LUA [8] a to z několika důvodů. LUA při spuštění umožňuje překlad skriptů do Byte kódu, což vede k rychlejší interpretaci kódu. Důležitá je také jednoduchost jazyka - jak po syntaktické stránce, tak i integrační. Naučení se základům jazyka LUA i jeho propojení s C++ programem je tak otázkou chvíle. Neopomenutelnou výhodou je i velikost knihovny potřebné k překladu a interpretaci skriptů, která se pohybuje kolem 100 Kb.

LUA komunikuje s C++ programem prostřednictvím speciálního společného zásobníku. Přes tento zásobník si jazyky mohou vyměňovat data. Datová výměna je však omezená jen na základní datové typy se kterými LUA pracuje – především čísla, řetězce a funkce (v LUA jde o datový typ). Přímá výměna složených datových typů tedy není možná.

Pokud z C++ programu potřebujeme zavolat LUA skript, musíme ho nejprve uložit na zásobník, vložit vstupní parametry a poté nechat skript pomocí LUA INTERPRETRU vykonat. Případné návratové parametry jsou vráceny opět na společný zásobník, odkud si je můžeme vyzvednout. Obdobně můžeme ze skriptu volat i C funkce. Popsanou situaci ilustruje obrázek níže.



Obr 4: Volání LUA skriptu z C++

6.3 Rozdělení skriptů

Navržený skriptovací systém lze rozdělit do několika samostatných částí. Každá část je jiného zaměření. Důsledkem toho je i lehce odlišný návrh a implementace těchto částí. Tím je mimo jiné umožněno lepších optimalizací a zrychlení zpracování skriptů.

6.3.1 Částicové skripty

Úlohou částicových skriptů je definovat vlastnosti nových částic. Částicové skripty jsou často vykonávané skripty a proto je jejich návrh přizpůsoben především efektivní implementaci. Tvorba nových částic ve skriptech probíhá prostřednictvím volání callback funkcí psaných v C.

Vytvářet lze konkrétně PARTICLE a EMITTOR částice. PARTICLE jsou koncové částice. Jsou jednodušší a rychlejší, mají však omezenější funkcionalitu. EMITTOR částicím lze navíc přiřazovat další částicové skripty, jejichž prostřednictvím mohou emitory

tvořit nové částice. Částice můžeme prostřednictvím skriptů také vzájemně spojovat pružinami.

6.3.2 Konfigurační skripty

Další kategorií skriptů jsou konfigurační skripty. Jde o jednoduché skripty, provádějící různé inicializace. Mezi tyto skripty patří především skripty definující řídící emitory, skripty registrující částicové skripty a skripty provádějící celkovou inicializaci aplikace.

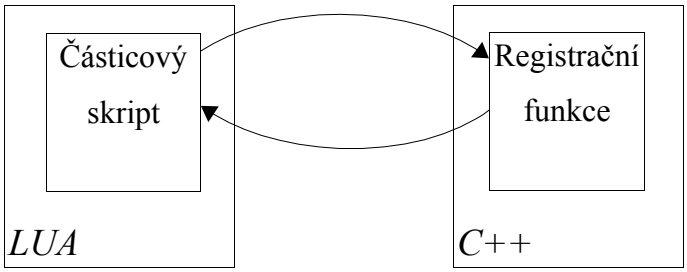
6.4 Zpracování částicových skriptů

V předchozích částech jsme ukázali, jak volání a zpracování skriptů v jazyce LUA probíhá a zavedli jsme několik různých kategorií skriptů, se kterými budeme pracovat. V této podkapitole se podrobněji podíváme na návrh a zpracování částicových skriptů.

V momentě zavolání částicového skriptu z C++ programu je na zásobník vložen příslušný skript a jeho vstupní parametry. Poté zpracuje skript LUA interpretr a proběhne návrat do C++ kódu. Součástí skriptu mohou být i předem registrované C funkce. Této funkcionality je využíváno v částicových skriptech k alokaci a uložení nových částic do dříve navržených datových struktur.

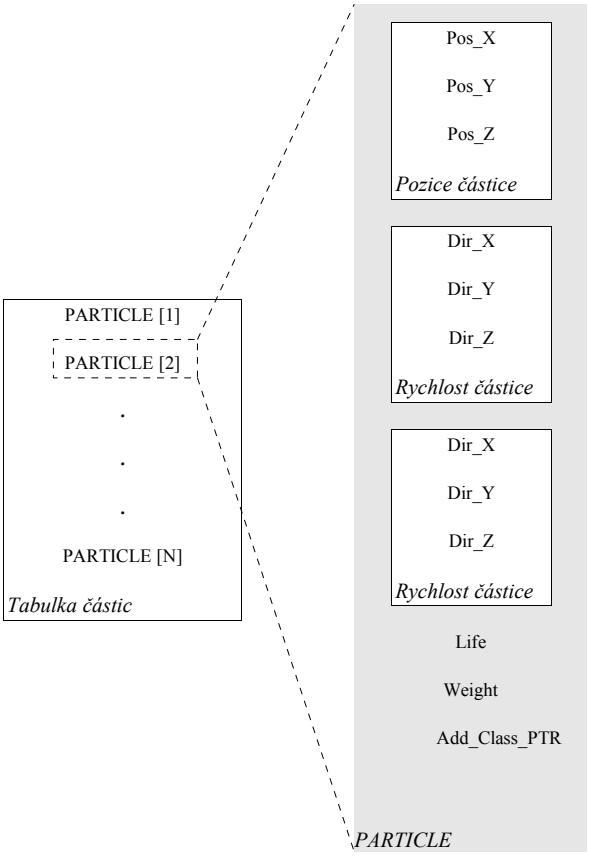
Zpracování částicových skriptů můžeme shrnout v následující postup

1) Ve skriptu definujeme vlastnosti částic.
2) Zavoláme ze skriptu registrační C callback funkci, přijímající tabulku atributů nových částic.
3) Tato funkce alokuje prostor pro nové částice v C++ části programu, zpracuje tabulku atributů, inicializuje nové částice a navrátí řízení do míst odkud byla zavolána.

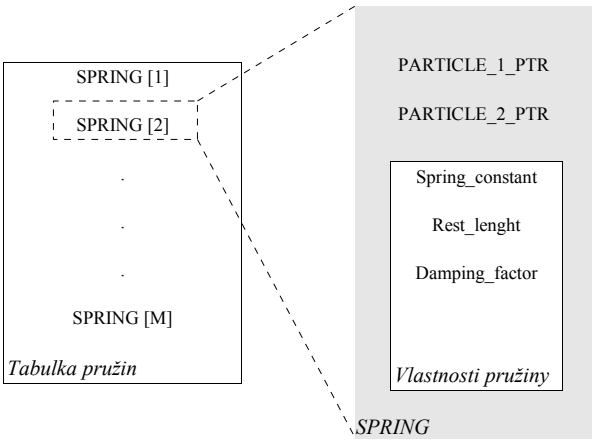


Callback funkce

6.4.1 Schéma zásobníku při zpracování částicových skriptů

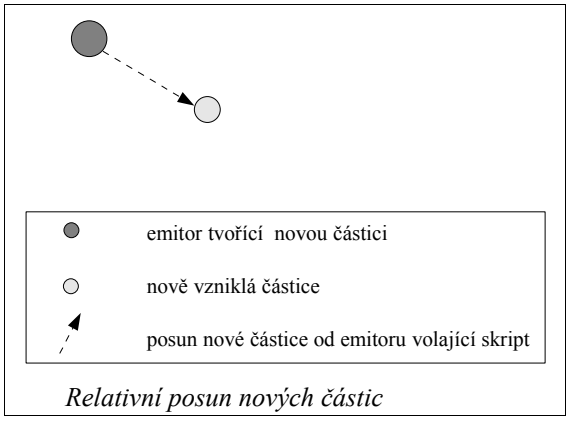


Reprezentace částice na zásobníku



Reprezentace pružiny na zásobníku

6.4.2 Význam parametrů

Parametr	Význam
pos_x, pos_y, pos_z	Určuje relativní posun částice na dané ose vůči emitoru, který skript zavolal. Částice tedy přebírá pozici svého předka, ke které jsou dále připočteny uvedené hodnoty.  *Relativní posun nových částic*
dir_x, dir_y, dir_z	Určuje relativní rozdíl rychlosti v daném směru vůči emitoru, který skript zavolal.
acc_x, acc_y, acc_z	Nastavuje absolutní zrychlení částice v daném směru. Částice nepřebírá po rodiči jeho zrychlení.
life	Nastavuje životnost částice. Po uplynutí této doby může být částice odstraněna .
weight	Nastavuje hmotnost částice. Pro rychlejší implementaci je vyžadována hodnota 1/hmotnost částice.
particle_1_ptr, particle_2_ptr	Ukazatele na částice které mají být vzájemně spojeny danou pružinou.
spring_constant	Nastavuje tuhost pružiny.
rest_length	Klidová délka pružiny
damping_factor	Útlum pružiny
scripts	Jde o tabulku s názvy částicových skriptů, jejími vstupními parametry a údaji o periodě s kterou mají být skripty volány. Tyto skripty se všemi uvedenými parametry jsou potom přiřazeny vytvářenému emitoru, který pak skripty může vo-

	lat. Syntaxe řetězce: `{{„nazev_skriptu1“,{vstupni_param1,…,vstupni_paramN}, perioda_volani1}, ...}`
add_class_ptr	Jde o C++ ukazatel na instanci potomka třídy ADD_DATA_CLASS, který má být přiřazen nové částici. Tento ukazatel může být získán voláním C++ callback funkce z LUA skriptu.

7 VIZUALIZACE ČÁSTIC

Doposud jsme se zabývali způsoby, jakými částice budeme vytvářet a simulovat. V této kapitole se podíváme na vlastní vizualizaci takových částic. Vizualizace částic je separovaná část kódu od předchozího návrhu. Lze ji proto dobře rozšiřovat nebo zcela změnit. K vizualizaci bylo použito grafické knihovny OpenGL [9]. Ta nám umožňuje využívat rychlých grafických karet pro zobrazování částic a renderovat tak potřebné množství částic bez výraznějšího zpomalení aplikace. Abychom však dokázali částice rychle zobrazovat, je potřeba přizpůsobit celkový návrh potřebám OpenGL.

7.1 Úvod k OpenGL

7.1.1 Co je OpenGL

Knihovna OpenGL je určená pro akcelerovaný rendering polygonální grafiky v třírozměrném prostoru. Jde o konečný stavový automat. OpenGL funguje na bázi klient-server. Klientem v tomto případě rozumíme část aplikace, která vydává zobrazovací příkazy. Server v podobě grafické karty tyto příkazy potom provádí.

7.1.2 Jak OpenGL zobrazuje

OpenGL pracuje s body v třírozměrném prostoru. Budeme je označovat jako vertexy. Těmto vertexům lze přiřazovat různé vlastnosti jako například barvu, průhlednost nebo texturové koordináty. Především je však lze vzájemně spojovat v polygony a zobrazovat je na výstup.

7.2 Návrh datových struktur pro OpenGL

Pro co nejlepší rendering částic potřebujeme navrhnout vhodné datové struktury, které nám umožní s částicemi pohodlně pracovat a současně je také rychle zobrazovat. Než tak učiníme, podívejme se krátce na způsoby, které OpenGL v tomto směru nabízí.

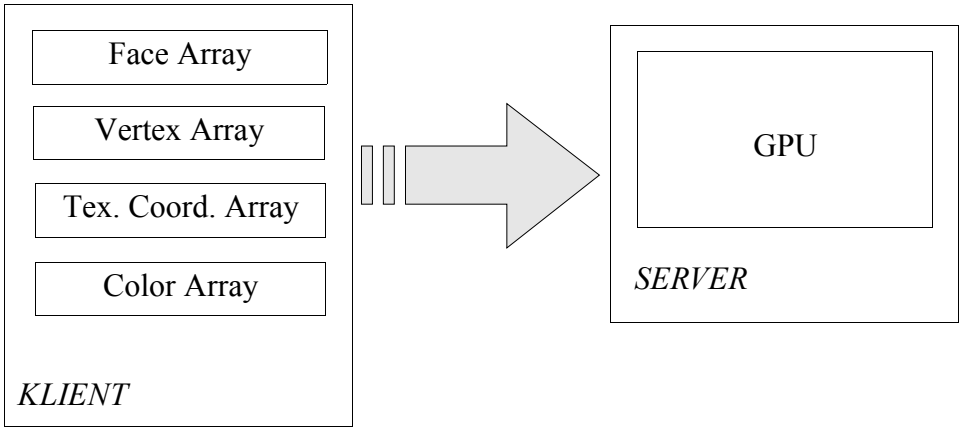
Data můžeme uchovávat na straně klienta nebo serveru. Uchováváme-li data na straně serveru, snižujeme tak objem dat posílaných mezi klientem a serverem při zobrazování jednotlivých snímků. Zároveň však k datům nemáme přímý přístup, protože se nenachází v operační paměti počítače, ale v paměti grafické karty.

My však budeme potřebovat data průběžně měnit a proto je pro nás vhodnější uchovávat data na straně klienta. Pro tyto situace OpenGL poskytuje metody tzv. Vertex

Array. Ta nám umožňuje data z klienta rychle přenášet na server, kde jsou následně zpracována.

7.2.1 Vertex Array metoda

Vertex Array metoda [9] a [10] vyžaduje uložení všech dat posílaných na grafický hardware v souvislých blocích paměti. Tyto bloky jsou na straně klienta a lze je tedy libovolně modifikovat. V době renderingu potom s tímto blokem pracuje i server.



7.2.2 Render manager

Když známe možnosti OpenGL, můžeme přejít k návrhu naší zobrazovací třídy pro vizualizaci částic. Třída bude zobrazovat částice podle jejich barvy, průhlednosti, velikosti, texturovacích koordinátů a přiřazených textur. Snadno najdeme i mnohé další vlastnosti, které by bylo možné přidat. Tyto speciálnější požadavky však necháme do případných rozšíření této třídy.

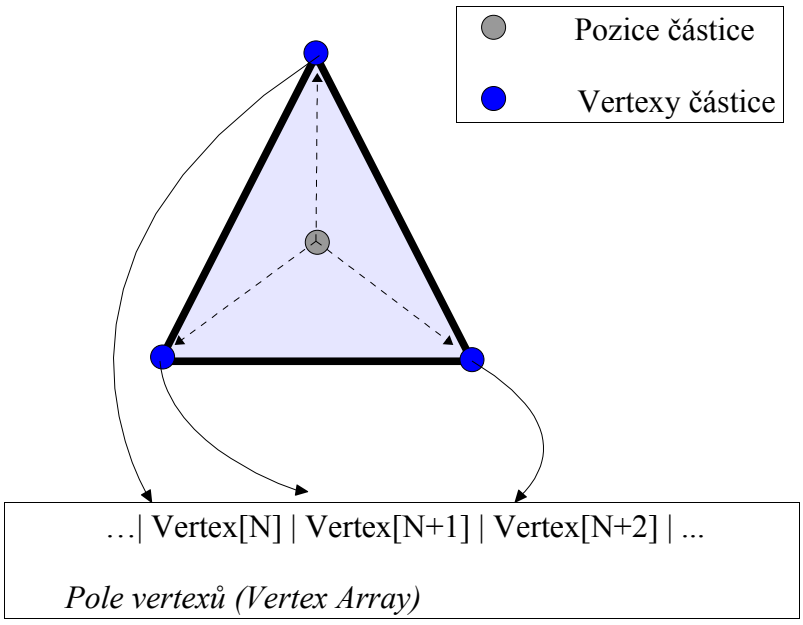
Veškerá data posílaná na GPU bude třída ukládat do před připravených bloků paměti, abychom mohli využít popsané Vertex Array metody. Částice budeme renderovat jako polygony. Doposud jsme však částice reprezentovali pouze jako body v Euklidovském prostoru. Vzniká proto potřeba tyto body převádět na polygony. OpenGL nejrychleji zobrazuje trojúhelníkové polygony a proto budeme používat právě je. Částice budeme převádět na trojúhelníky a otáčet je směrem k pozorovateli. Tím jednak lépe vyjádříme podstatu

částic coby malých uniformních těles a jednak tím i opticky zvýšíme počet částic. U simulací si tak vystačíme i s menším počtem částic, aniž by scéna působila prázdně.

7.2.3 Billboarding

Otáčení polygonů směrem ke kameře souhrnně označujeme jako billboarding [11]. My ho budeme využívat hned k několika úkonům současně. Během billboardingu budeme převádět pozice částic na vertexy představující vrcholy polygonu. Tyto vertexy budeme rovnou ukládat do před alokovaného bloku v paměti, tak jak to ukazuje obrázek níže.

V době renderingu bude potom tento blok paměti spolu s dalšími bloky obsahujícími texturovací koordináty, barvy vertexů a indexy vertexů využívá grafická karta jako zdroj dat pro zobrazování částic, tak jak to vyžaduje popsaná Vertex Array metoda.



8 ZÁVĚR

8.1 Shrnutí výsledků práce

V úvodní kapitole jsme vytyčili cíle této práce. Podívejme se proto v krátkosti nakolik se je povedlo splnit. Podařilo se nám navrhnout a implementovat funkční skriptovací systém pro definici částic. Tento systém umožňuje jednoduše a intuitivně popisovat vlastnosti částic. Zároveň poskytuje i potřebnou rychlost pro práci s větším množstvím částic. Použitý fyzikální model tyto vytvořené částice umožňuje simulovat. Implementovaná Eulerova metodu funguje pro účely většiny simulací uspokojivě. Je velmi rychlá a zvládá tak počítat mnoho částic bez výraznějšího zpomalení. Pouze pro menší a preciznější simulace se občas ukazuje jako numericky nepřesná. Nakonec, prostřednictvím jednoduché grafické nadstavby, lze částice zobrazovat. Můžeme proto říci, že cílů jsme úspěšně dosáhli.

8.2 Směr další práce

Tím se dostáváme k otázce případných budoucích rozšíření stávající aplikace. Zde se domnívám, že by nemělo být problémem přidávat novou funkcionalitu a od počátku tak byl projekt koncipován. Většina důležitých částí je proto navržena a implementována jako oddělené moduly komunikující spolu přes společné rozhraní. Fyzikální simulátor, skriptování i renderer je tak možné kdykoliv změnit, nahradit nebo rozšířit s minimálním dopadem na ostatní komponenty programu.

Jak bylo naznačeno v předchozí části, byl projekt navržen k dalšímu rozšiřování. Při svém rozsahu a tématické variabilitě práce snad ani nelze očekávat dodání dokonale komplexního systému pro tvorbu, simulaci a vizualizaci částic ve všech jeho ohledech. Za důležité proto považuji především možnost rozšiřovat aplikaci podle individuálních potřeb.

Větší pozornost by si například zasloužila lepší podpora grafických karet. Ty jsou dnes velmi výkoné a pro svou nízkou pořizovací cenu také lehce dostupné. Staly se proto prakticky nedílnou součástí každého nového počítače. Mohli bychom je využít zejména ke zvýšení vizuální kvality částic použitím pokročilých shader modelů nebo provádět část fyzikální simulace prostřednictvím grafické karty.

9 POUŽITÁ LITERATURA

[1] http://www.cg.tuwien.ac.at/courses/Animation/I_Physically_Based.pdf

[2] http://mathonline.fme.vutbr.cz/Mocninne-a-Taylorovy-rady/sc-39-sr-1-a-54/default.aspx

[3] http://www.fce.vutbr.cz/studium/materialy/Dynsys/kap7/kap7.htm

[4] http://kfe.fjfi.cvut.cz/~limpouch/numet/foluvoda.pdf

[5] David M. Bourg: Physics for Game Developers

[6] Brdička, Hladík: Teoretická mechanika

[7] Richard Fitzpatrick: Classical Mechanics

[8] http://www.lua.org/manual/5.1/manual.html

[9] http://www.opengl.org/documentation/specs/

[10] Shreiner D., Woo M., Neider J., Davis T.: OpenGL Průvodce programátora

[11] http://www.lighthouse3d.com/opengl/billboarding/billboardingtut.pdf

[12] Zara J., Benes B., Felkel P., Sochor J.: Moderni pocitacova grafika

[13] Andrew Witkin: Physically Based Modeling: Principles and Practice Particle System Dynamics

[14] Wiliam T. Reeves: Particle systems—a technique for modeling a class of fuzzy objects

[15] Sebastian Sylvan: Particle System Simulation and Rendering on the Xbox 360 GPU

[16] http://ocw.mit.edu/NR/rdonlyres/Architecture/4-491Fall-2004/BD64143E-1DC1-4813-8741-20AD3A981C6C/0/ln1.pdf

[17] http://unity3d.com/support/documentation/Manual/Particle%20Systems.html

[18] http://www.wikipedia.org/

10 PŘÍLOHA

SEZNAM PŘÍLOH

10.1 Tvorba částicového efektu krok za krokem

Nejprve je potřeba zaregistrovat částicové skripty, které bude efekt používat. Tuto registraci provedeme prostřednictvím speciálního skriptu. Tento skript musí vracet tabulku všech používaných částicových skriptů. Název tohoto skriptu je pevně daný, musí se vyskytovat ve stejném souboru jako skripty, které inicializuje a tento skript je implicitně zavolán v době inicializace nového efektu.

```
function log()
        return {{{"emittors1"}, {"particles"}}}
end
```

Dále je potřeba definovat inicializační skript. Tento skript bude zavolán pouze jednou a to při inicializaci nového efektu. Skript musí inicializovat řídící (tj. hierarchicky nejvyšší) emitor.

```
NULL=0;

function init_main_script()
   emittor_array={};

   pos={0,0,0};
   dir={0,0,0};
   acc={0,0,0};
   life=32000;
   weight=1.0;
   args={-1,1};
   period=80;

   emittor_array[1]={pos,dir,acc,life,weight, NULL, {{"emittors1",args,period}}};
   return {emittor_array};
end
```

Vytvořili jsme hlavní emitor na pozici (0,0,0) s počáteční rychlostí (0,0,0), zrychlením (0,0,0), životností 32000 a hmotností 1.0. Emitor neobsahuje žádný ukazatel na dodatečná data (nastavili jsme ho na NULL) a volat bude jediný skript s názvem „**`emittors1`**“ se vstupními parametry -1,1 a periodou 80.

Nyní tento skript definujeme.

```
function emittors1( arg1, arg2, calling_emittor)

   emittor_array={};

   emittor_array[1]={{arg1, arg2, arg1+arg2},{0,70,0},{0,0,0},600,1.00,
               get_ptr_to_new_instance(1,2,3,4),  {{"particles",{},2}};

   insert_new_emittors({emittor_array},calling_emittor);
end
```

Skript přijímá tři vstupní parametry. Poslední parametr je vždy ukazatel na emitor, který skript zavolal a jde o programem automaticky přidávaný parametr.

Ve skriptu vytváříme vždy jeden emitor na pozici (arg1, arg2, arg1+arg2), rychlostí (0,70,0), zrychlením (0,0,0), životností 600 a hmotností 1.0. Emitoru dále přiřadíme adresu na přídavná data. Tuto adresu vrací námi definovaná C callback funkce **`get_ptr_to_new_instance`**. Funkce musí být předem zaregistrována, aby ji bylo možné ze skriptu volat. Funkcí vrácená adresa odkazuje na objekt, který se stává nedílnou součástí částice. Tento objekt je možné využívat k dodatečným informacím o částici, které nejsou standardně podporovány. Pokud částice žádná přídavná data nepotřebuje můžeme ukazatel nastavit jednoduše na 0. Nakonec vytvářenému emitoru přiřadíme skript „**particles**“, který nemá žádné vstupní parametry (s vyjímkou automaticky dodávané adresy rodiče) a je volán s periodou 2. Definované částice nakonec přidáme do C++ programu příkazem **`insert_new_emittors`**

Zbývá definovat „**`particles`**“ skript

```
function particles(calling_emittor)

   particle_array={};

   for index=1,10 do
   particle_array[index]={{0,0,0},{index, index*2, index*3},{0,0,0},100,1.00,
                          get_ptr_to_new_instance(5,6,7,8)};
   end

   insert_new_particles({particle_array}, calling_emittor);
end
```

Skript generuje 10 částic s pozicí (0,0,0), rychlostí (index, index*2, index*3), zrychlením (0,0,0), životností 100, hmotností 1.0 a ukazatelem na vrácenou adresu C++ funkce **`get_ptr_to_new_instance`**.

Pro úplnost dodejme ještě ukázkový příklad, jak by mohla vypadat C funkce alokující prostor pro přídavné atributy částic.

```
LUA_FUNC get_ptr_to_new_instance(lua_State* L)
{
   double p1=lua_tonumber(L,1);
   double p2=lua_tonumber(L,2);
   double p3=lua_tonumber(L,3);
   double p4=lua_tonumber(L,4);
   CLASS1* new_instance=    new CLASS1(p1,p2,p3,p4);

   lua_pushnumber(L,(unsigned int) new_instance);     //ulozim na zasobnik ukazatel na objekt
   return 1;
}
```

Poznámka: Jedná se pouze o smyšlený příklad popisující tvorbu nového efektu. Skutečné funkční efekty je možné prohlédnout ve složce `/ParticlesBIN/LUA_SKRIPTY` na přiloženém CD.

10.2 Třída PARTICLE_EFFECT

PARTICLE_EFFECT je předdefinovaná třída pro univerzální tvorbu částicových efektů. K popisu částic používá systém skriptování a k vizualizaci navrženou třídu Render-Manager. Vizuální popis částice je uchováván ve speciálním objektu, který spolu s každou částicí inicializujeme. Tento objekt pracuje s následující sadou parametrů.

PARAMETR	VÝZNAM	ROZSAH HODNOT
size	Určuje počáteční velikost částice v době vzniku částice.	double
size_offset	Určuje změnu velikosti v průběhu simulace. Původní velikost částice je v každém volání metody render() inkrementována o tento parametr. Tím lze docílit toho, že se částice v průběhu simulace zvětšuje (kladné hodnoty parametru), nebo naopak zmenšuje (záporné hodnoty parametru).	double
color[0-3]	Určuje odstín vzniklé částice. Kde color[0] reprezentuje červenou, color[1] zelenou a color[2] modrou složku barvy.	float [0,1]
color[4]	Určuje počáteční průhlednost částice.	float [0,1]
alpha offset	Změna průhlednosti v průběhu simulace. Podobně jako size_offset i zde je tento parametr průběžně přičítán k color[4].	float [-1,1]
tex.coord[0,2,4]	Texturovací koordináty na ose X pro vertexy polygonu.	float [0,1]
tex.coord[1,3,5]	Texturovací koordináty na ose Y pro vertexy polygonu.	float [0,1]

Objekt dále obsahuje několik dalších atributů nesouvisejících s grafickou podobou částice.

PARAMETR	VÝZNAM	ROZSAH HODNOT
apply_gravity	Informace, zda má být na částici aplikována gravitační síla.	boolean
apply_spring force	Informace, zda mají být na částici aplikovány síly pružin, se kterými je částice spojena.	boolean
pickable	Informace, zda má být částice registrována v objektu PICK_UP.	boolean

Uvedené parametry je možné nastavovat přímo v částicových skriptech prostřednictvím callback funkcí **`get_ptr_to_new_instance_PARTICLE_EFFECT`** a **`get_ptr_to_new_instance_PARTICLE_EFFECT_2`**, které jsou detailněji popsány v programátorské dokumentaci

Objekt typu PARTICLE_EFFECT můžeme vytvořit buď přímo v C++ programu nebo i v konfiguračním skriptu příkazem **`create_new_PARTICLE_EFFECT`**.

Celkové zapojení třídy můžeme shrnout následujícím schématem.

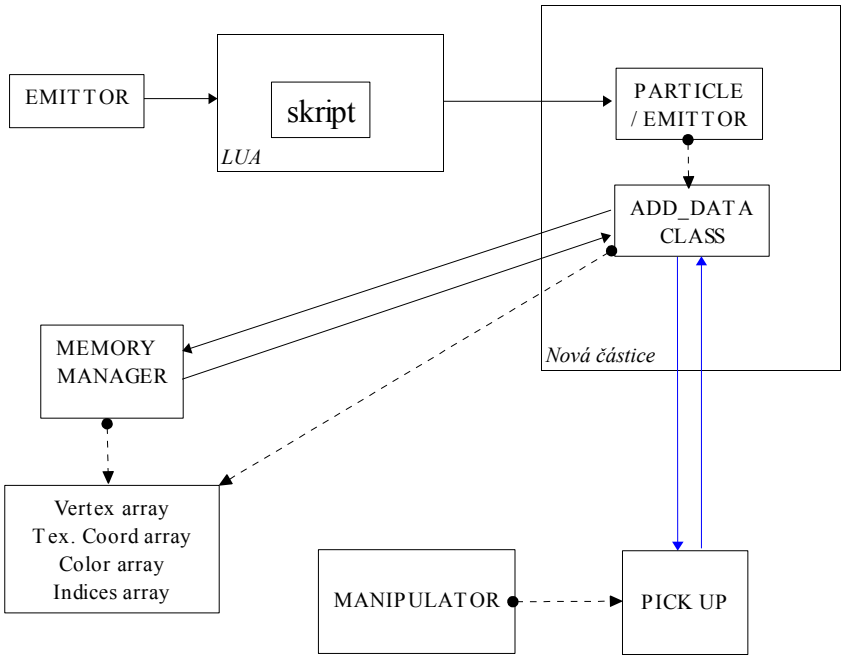
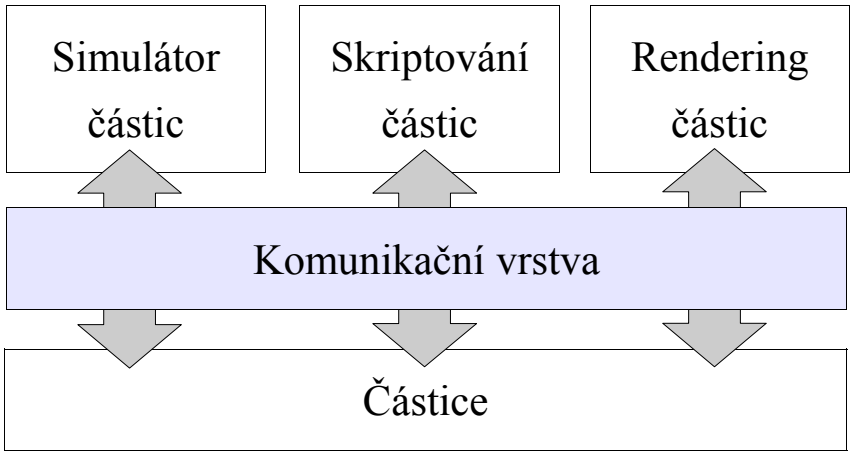


Schéma propojení v PARTICLE EFFECT

KOMENTÁŘ

1. Emitory obsažené v objektu PARTICLE_EFFECT volají LUA skripty.
2. Uvnitř skriptu se definují vlastnosti nových částic a volají se registrované C++ call-back funkce (dříve popsaná get_ptr_to_new_instance_PARTICLE_EFFECT).
3. Callback funkce získá od interního správce paměti místo v před vytvořených buffe-rech, kam nová částice ukládá své texturovací koordináty, barvu a vertexy. S těmito buffery potom pracuje grafická karta při vlastním renderu částic.
4. V případě, že částice má být PICKABLE, probíhá navíc registrace v objektu PICK UP.
5. Prostřednictvím objektu MANIPULATOR nakonec probíhá případná interakce s vybranými objekty.

10.3 Diagram programu



Jádro programu

10.4 Anglická terminologie

- particle systems - částicové systémy
- rendering - vizualizace počítačových dat
- real-time - výpočty prováděné za běhu v reálném čase (daný výpočet by měl být spočitatelný alespoň 25x za sekundu aby nedocházelo k viditelným neplynulostem)
- shader - jednoduchý program zpracovávaný na grafické kartě
- callback funkce - nízkoúrovňové funkce, které jsou volány z vyšší řídící vrstvy, dočasně přebírají řízení programu, po vykonání své úlohy navracejí řízení vyšší úrovní do míst odkud byly zavolány
- GPU (Graphic Processing Unit) - grafický procesor; jde o specializovaný čip umístěný na grafické kartě provádějící specializované výpočty
- vertex - bod v třírozměrném prostoru
- face - mnohoúhelník definovaný posloupností vertexů
- billboarding – otáčení polygonů směrem ke kameře

10.5 Obsah CD

Důležitým doplňkem této práce je přikládané CD. Na tomto CD jsou obsaženy zdrojové kódy projektu, programátorská i uživatelská dokumentace a především spustitelný program s ukázkami několika simulací. Obsah adresářů na CD je následovný:

- **`/ParticlesBIN`** – Spustitelný program s ukázkami. Bližší informace o jednotlivých simulacích, jejich realizaci a významu se lze dočíst v souboru „Ukazkove_priklady_komentar.pdf“ umístěný v téže složce.
- **`/ParticlesSRC`** – Obsahuje zdrojové kódy k projektu.
- **`/ParticlesDOC`**
  - **`/ParticlesDOC/PRG`** – Složka s programátorskou dokumentací, která popisuje implementaci a návrh jednotlivých částí programu.
  - /**`ParticlesDOC/USR -`** Uživatelská dokumentace obsahuje popis ovládaní přiložené aplikace.
  - **`/ParticlesDOC/GEN -`** Složka s automaticky generovanou dokumentací.